Intrusion Detection Force:
An Infrastructure for Internet-Scale Intrusion Detection

Lawrence Teo[1,2] Yuliang Zheng[1,2] Gail-Joon Ahn[1]

[1] *Laboratory of Information Integration,*
Security and Privacy (LIISP)
University of North Carolina at Charlotte,
9201 University City Blvd,
Charlotte, NC 28223, USA.

[2] *Calyptix Security Corporation*
P.O. Box 561508,
Charlotte, NC 28256, USA.
http://www.calyptix.com/

$\{$*lcteo,yzheng,gahn*$\}$*@uncc.edu*

Abstract

Intrusion Detection Systems (IDSs) are usually deployed within the confines of an organization. There is usually no exchange of information between an IDS in one organization with those in other organizations. The effectiveness of IDSs at detecting present-day sophisticated attacks would increase significantly if there are inter-organizational communication and sharing of information among IDSs. We envision a global Internet-scale defense infrastructure, which we call the Intrusion Detection Force (IDF), that would protect organizations and defend the Internet as a whole. This paper provides a blueprint of the IDF, where we discuss the requirements to deploy such an infrastructure, and describe its architecture and design in terms of its basic building blocks and major components. We also describe a few applications of the IDF architecture, and provide a small experimental prototype that we are currently extending as part of our vision to implement the full IDF infrastructure.

Keywords: Intrusion Detection, Intrusion Detection Force, Survivability, Internet Scale, Information Sharing, Infrastructure, Recovery, Fault Tolerance.

1. Introduction

Since its inception more than 20 years ago, the field of intrusion detection has been growing rapidly. Early intrusion detection systems (IDSs) catered only for a single host or at most, a small network. As networks expanded and organizations grew, there was clearly a need for large-scale distributed intrusion detection. This led to the emergence of distributed IDSs such as NADIR [12], Distributed Intrusion Detection System (DIDS) [20], GrIDS [7], and AAFID [4]. Commercial IDSs have also adopted the distributed data collection and processing paradigm.

Although these IDSs handled distributed intrusion detection, it can be observed that they concentrated mainly on handling the task of intrusion detection only within the one organization that they are deployed in. An IDS in Organization A does not communicate at all with the IDS in Organization B. Without inter-organizational information sharing, the potential of the IDSs and intelligence-gathering ability of these organizations become severely limited. For example, a setup such as this makes it difficult to detect distributed and stealthy attacks that span across the Internet, such as distributed denial of service (DDoS) attacks.

The current infrastructure of the Internet is another factor limiting the ability of organizations to conduct better attack detection and prevention. Since the TCP/IP protocol suite was not designed with security in mind [6], it is infeasible to rely on it as a foundation for security.

To address these problems, it is evident that there needs to be a form of information sharing among various security technologies in different organizations. However, the scope of information shared must not be limited to just a few organizations; for information sharing to be effective, it needs to span across a very large scale, such as through the entire Internet. The insecurity of TCP/IP calls for a logical, virtual and secure layer on top of the current Internet infrastructure.

To enable this solution, we envision the creation of a virtual infrastructure that would allow Internet-scale intrusion detection. We would like to introduce the term "Intrusion Detection Force", or IDF, to describe this infrastructure. The IDF would be used by millions of hosts throughout the Internet, spanning different organizations, countries, and continents.

Why an "Intrusion Detection Force"? To explain the rationale and philosophy behind an intrusion detection force, we wish to draw a parallel between the introduction of IDF and the historical evolution of conventional military forces. In the early history of humankind, there were small tribes of similar people. To defend themselves, tribes typically live in their own villages with boundaries to protect themselves against other tribes. As human relations improved, various tribes began to live together and villages gradually grew into towns. As larger numbers of people live together, the defense force also increased in size and adopted a more organized structure. Towns later emerged into cities, and cities were later grouped into nations with their own large defense forces. We can also see from history that nations grouped together to become larger nations. Currently, it can also be seen that nations with similar interests are forming international coalitions with cooperating defense forces. Another interesting observation is that the modern military consists of many forces, such as the Army, Navy, and Air Force, which perform different tasks in different areas, but the overall goal is still to defend a nation.

Like the military forces, the IDF infrastructure we propose will also have entities and groups of entities which span the Internet and work in different areas, but the overall goal is to defend organizations and the Internet as a whole. We are very much aware that the task of building such a large infrastructure is enormous. There are numerous issues to address and problems to solve, such as the design, protocols, scalability, target systems, and malicious attacks on the system itself. Due to the enormity of the task, it would be near-impossible to produce actual results in the short term. Therefore, the aim of this paper is not to present full and final results, but to provide a high-level blueprint and roadmap of the IDF for our long-term future work (having said this, we will discuss some preliminary results from our earlier work [23] in Section 5). We believe that this blueprint is extremely important as it outlines the design and architecture of a potential global Internet-scale defense infrastructure. More specifically, we will present the conceptual framework, architecture, design, and components of the IDF. We will discuss the realistic requirements and practical issues that need to be addressed in order to deploy the Intrusion Detection Force on the Internet.

In summary, we propose an Internet-scale Intrusion Detection Force, which is a virtual infrastructure on top of the current Internet that enables secure information sharing and intelligent data analysis and response. The main objective of the IDF is to defend organizations and protect the Internet as a whole, in a way that is not possible before. Throughout the paper, we will explain the IDF architecture and its applications. The concepts discussed in the paper were designed and formulated through careful consideration of practical and realistic deployment issues.

The rest of this paper is organized as follows: Section 2 presents the main requirements of the IDF infrastructure. Section 3 presents the architecture and design, which aims at fulfilling the requirements. It also describes the basic entities of the architecture as well as a discussion of the hierarchical model used. Section 4 shows components built from the entities discussed in Section 3 , along with a description of their functions, how they relate to the IDF entities, and their applications. Section 5 describes our current implementation with a description of our research prototype and tests performed. Related work is presented in Section 6, before the conclusion in Section 7.

2. Requirements

The four key requirements of the IDF are information sharing, scalability, security, and scalability. This section discusses these requirements in detail. Other requirements are explained later in the section.

2.1. Information sharing

Information sharing is the "glue" and core strength of the IDF. Information sharing is the active exchange of information among the IDF community that will benefit each member in the IDF. The information shared is analyzed and used to protect and defend the organizations themselves. This is different from the behavior of the current generation of IDSs, which mainly confines all information inside the organization itself.

Our emphasis on inter-organizational information sharing is reminiscent of the physical world. Following the terrorist attacks in the United States on September 11, 2001, the Bush administration proposed the Department of Homeland Security, which acts as an information clearinghouse for various law enforcement and intelligence agencies. This allows information gathered from intelligence activities to be analyzed and correlated in a way that could not have been done should the agencies continue to confine information within themselves (as was done in the past). In the intrusion detection community, we can already see efforts heading in this direction. For example, there is an IETF Intrusion Detection Working Group (IDWG) [13] which is developing the Intrusion Detection Message Exchange Format (IDMEF) to provide a standardized message format for exchanging messages between different IDSs (it should be noted that the IDWG was already present prior to the events of September 11).

In the context of the IDF, it should be noted that the term "information" does not refer to proprietary information of companies and organizations; rather, it refers to network traffic data and generic data collected from hosts that are used by the IDF for analysis. In reality, however, organizations will still be concerned about the exchange of information, even if it is clearly shown that it is not proprietary. This is understandable because it is only natural for organizations to be concerned about the privacy of their data. However, certain protocols may contain proprietary information that is sent in the clear over the network. For example, it cannot be avoided if an insecure telnet connection shows some amount of confidential information over the network in the clear (of course, an organization that still uses telnet in this age should seriously revamp and enforce its security policy).

This problem can be addressed in two fronts using technology and policy. Firstly, the technology has to be sophisticated enough to intelligently filter outgoing information by blinding out private information while preserving other information needed for analysis. The technology has to decide which parts of the information need to be blinded, and which do not.

The second approach is to allow organizations to define and customize the information sharing policy. This allows organizations to determine the level and the amount of information to be shared with other organizations. It is also important to provide users with pre-defined policies for them to choose. We should not assume that all users are advanced users who fully understand security and the consequences of information sharing. However, this is a human interface issue which we will not discuss in detail in this paper.

2.2. Scalability

The Internet consists of millions of inter-connected hosts. In order for the IDF to scale to the size of the Internet, it has to cater to the needs of millions of hosts. Having said this, it is not realistic to assume an overnight deployment of the IDF throughout the entire Internet. Since deployment will definitely be conducted in phases over a significant period of time, the IDF architecture has to be designed to scale and grow from small networks to large networks. The types of entities in the IDF architecture and the decisions on how they can communicate with each other have to be carefully evaluated.

We can learn a few strategies for achieving scalability from other tried-and-true technologies which have been proven to scale in this manner. Consider the Internet Protocol (IP), which is known to be highly scalable and has the ability to run over almost anything. Some of the strategies we can learn from IP are to make minimal assumptions about the underlying network, and to use global, unique addressing methods (a notion which is also used in Internet-scale research operating systems, such as Chord [22]). We can also learn from the problems of IP, such as IPv4's inability to scale past 4 million nodes, and the solutions proposed in IPv6.

2.3. Security

The IDF's role in exchanging information between different organizations brings with it the responsibility of securing these information exchanges. Since the IDF will be deployed on systems and networks which are untrusted and which may have questionable reliability, we will make two conservative assumptions that will affect the security requirements of the IDF:

- **Assumption 1:** The IDF is operating in a hostile and non-trusting environment.
- **Assumption 2:** The IDF is operating on an unreliable underlying network.

We will now discuss IDF security requirements in terms of the three traditional aims of security: Confidentiality, integrity, availability.

The nature of the information exchanged by the IDF between organizations can be used to gather intelligence about the organizations for malicious purposes. Therefore, it is clearly evident that we have to keep IDF information exchanges totally confidential. This defends against attacks such as eavesdropping and other malicious information-gathering attacks. Securing these information exchanges can be done using encryption, and one strategy to allow encrypted information exchanges for different organizations is to use a form of public key infrastructure (PKI).

The IDF would undoubtedly be a potential target for attackers, given the value of the assets it is designed to protect. Attackers would definitely be interested in modifying or spoofing the messages exchanged to confuse the system for illegal purposes. It is important that the integrity of messages exchanged be protected against unauthorized modification.

The IDF also has to ensure the availability of its services to organizations. However, due to the hostile environment of its deployment, this is not always possible. There are two types of incidents that can make systems unavailable: the first is the typical distributed denial-of-service attack. Although the IDF is designed to address such attacks, it may not always be possible to totally prevent them. The other category of incident that can render systems unavailable are accidents. Nodes may suddenly get disconnected due to physical problems or natural disasters. The other key requirement of survivability addresses these issues.

In order to provide maximum security of the IDF itself, it is important that the IDF architecture is hardened. Use of cryptography, authentication, and authorization schemes for the IDF to achieve this requirement must be carefully studied and tested. Implementation of the IDF architecture should be done using well-developed processes and secure programming practices.

2.4. Survivability

The fourth key requirement, survivability, is related to security, but it is distinct and important enough to warrant a separate section for its discussion. Survivability is defined [9] as "the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks, failures, or accidents." Survivability is vital because no system, no matter how well designed and implemented, is totally immune to failures. Survivability is even more important in the IDF because the IDF will be deployed in a highly distributed manner in potentially hostile environments throughout the Internet. The aim of survivability is not to prevent failures, but to ensure that even if the system fails, it will do so in a fail-safe manner.

One approach to achieve survivability in the IDF is to use mechanisms like application-level fault tolerance [5, 10, 11]. This is particularly apt to be used because we are not making the assumption that the underlying network on which the IDF runs is sufficiently fault tolerant. Thus, application-level fault tolerance can be used to complement existing system-level fault tolerance mechanisms.

2.5. Other requirements

Apart from the key requirements, there are other smaller requirements that the IDF has to fulfill. These requirements are interoperability, extensibility, and the need to achieve a balance between usability and security. These requirements are explained as follows.

2.5.1. Interoperability. A typical network environment of today consists of heterogeneous systems. The types of network, operating systems, hardware platforms, security solutions, and so on are very likely to be different. Legacy systems further complicate the situation. There are a few ways to deploy the IDF in such an environment. The first option is to enforce standardization of technologies on the user. By this, we mean that we will force the user to use only one type or a selected set of allowable technologies. In practice, this is highly undesirable, since users are more likely to stay with technologies that have previously worked for them. Furthermore, there may be cost and resource constraints that may prevent them from adopting new standards. In other words, it is unlikely that organizations would replace their current firewalls and intrusion detection systems unless the replacement technologies provide significantly more value than the current ones.

One strategy to address this is to integrate the IDF with whatever technologies that are currently in use in the organization. It is important that the integration solution be platform- and network-independent. This is necessary to allow deployment of the IDF into any potential heterogeneous environment in a typical organization.

2.5.2. Extensibility. With the IDF infrastructure in place, it is likely that there will be new potential applications that can take advantage of the infrastructure. To allow the development of future applications of the architecture, the IDF must be designed to be extensible and provide a secure platform for developers to design and implement new applications.

2.5.3. Balance between usability and security. Lastly, there needs to be a balance between usability and security. While the IDF can be designed to be secure and hardened, it is of limited use if it does not complement an organization's existing policies and practices. It also needs to be usable by both novice and advanced users. This last requirement can be addressed by designing an effective user interface and providing adequate training for users.

3. Architecture and design

We have presented the requirements of the IDF, and we will now describe its architecture and design. First, we present the key design decisions made to fulfill the requirements discussed in the previous section. Then, we explain the hierarchical model of the IDF architecture. This is followed by a discussion of the entities in the IDF architecture.

3.1. Design decisions

In order to accomodate the requirements stated in the previous section, we have carefully made a set of design decisions which we believe would best fit the IDF. It should be noted that these design decisions refer not so much to a specific intrusion detection engine itself, but rather the high-level IDF infrastructure which allows various intrusion detection engines to communicate. Another way to phrase it is that the IDF has entities which can wrap around other already-available intrusion detection engines. The following list describes our design decisions and justification on the aspects of detection, response, audit sources, interoperability, data collection, data processing, and systems security in the IDF:

1. **Detection must be done in real time.** In order to achieve high-speed response and proactively react to intrusions, it is evident that detection must be done in real time and not in batch mode.

2. **Response can be either passive or active.** The type of response can either be passive or active, depending on the security policy defined by the administrator.

Passive here means that the events will be stored in a log, while active means that the events will trigger some actions (such as attack countermeasures) which will respond accordingly to the event. Both modes are supported because it would not be practical to force the user into just one mode; one of the requirements of the IDF is to be highly interoperable, which also implies flexibility to adjust to various security policies. Response must also be high-speed and timely to react to events, regardless of whether the active or passive mode is chosen.

3. **The audit sources must be taken from both the host and network.** The IDF spans the entire Internet and runs on a variety of hosts and networks. Since attacks can be launched from and against both hosts and networks, the IDF's capabilities would be limited if the audit sources are confined to either host or network only. Unlike a regular intrusion detection engine, the IDF has to work with a very large scale and gather as much data as possible for analysis.

4. **The IDF needs to have a high degree of interoperability.** The IDF will be used in a heterogeneous environment. It needs to be highly interoperable with various operating systems, networks, machine architectures, and other security solutions.

5. **Data collection must be done in a distributed manner.** Since the IDF's target environment is extremely large and may consist of various heterogeneous systems, data collection must be performed in a distributed manner.

6. **Data processing must be done in a distributed manner.** Two of the requirements stated in the previous section are security and survivability. In order for the IDF to protect the organizations, the data collected from them must be available for analysis. Even if the systems that host them do fail, they must do so in a fail-safe manner. To accomplish this, data has to be replicated so that they can be retrieved from elsewhere even if the systems that host them are no longer available.

7. **The security of the IDF itself must be high.** This statement means that the IDF itself must be able to withstand hostile attack against its architecture. It must be resistant to tampering, even from the hosts it is running on. The security of an IDS itself is one area that has not been addressed by many other IDSs [3].

3.2. Hierarchical model

Before we commence a discussion of the entities in the IDF, it is important to discuss how the entities should be arranged in the architecture. Since the IDF is meant to be deployed on a very large scale, the entities must be arranged according to some form of hierarchy to ensure scalability.

The decision on the number of hierarchical levels in the IDF architecture is critical. This number will affect the feasibility of the IDF to be practically deployed, and therefore must be carefully considered. There are a few approaches we can take to determine the number of levels required to fulfill our main requirements of information sharing, scalability, security, and survivability.

The first approach we can use is to have a multi-level hierarchy, such as a hierarchy of four or five levels, or up to a certain large number of levels. This is somewhat like the Internet's Domain Name System (DNS) model [16]. An architecture with a hierarchy such as this would be able to compartmentalize the shared information to various defined areas. While this is a good property of this model, there are problems with it as well. As the network grows, the number of levels become complicated and rigid. There is also the problem of deciding what to do with the information stored at each level; in other words, this model becomes increasingly difficult to implement as the network grows. It becomes hard to predict problems that may occur if there is a large hierarchy of entities to consider. Also, an attack on a high-level entity may affect the entities below it.

At the other end of the spectrum, we can employ a totally distributed architecture with no hierarchy. This means that each entity in the system is an actual peer to another entity, where they share equal capabilities and rights. This model enables all entities to be implemented in a uniform manner, thus achieving simplicity in implementation. Furthermore, this model facilitates easy information sharing. Since all entities have equal rights, they can easily duplicate information for redundancy purposes and reduce the impact of an attack that shuts down a few entities. This model can be compared to the peer-to-peer filesharing systems of today. Interestingly, this model has some of the same disadvantages that we discussed with the multi-level hierarchical model. Again, as the network grows, potential problems become difficult to predict due to the anarchic nature of the network. The network becomes difficult to manage, and there is a danger of losing control.

For the purposes of the IDF, we propose the use of a hybrid model that combines the advantages of both the multi-level hierarchical model and the totally distributed model. We believe that a two-level hierarchy provides the optimal combination of both models. This model achieves simplicity since there are only two types of entities to implement. To achieve scalability and security, the entities will be allowed to link themselves to one another, somewhat like the distributed model discussed earlier. We will now present the entities involved in the IDF and how they relate to this hierarchical model.

Table 1. IDF entities: Names, definitions, and functions

Name	Definition and Function
Node	A node is a host which is running an IDF software agent. A node shares information with other nodes.
Collective	A collection of nodes. Nodes in a collective forward information to each other in order to achieve resiliency and support availability (security) and survivability.
Supernode	A supernode is a special node that provides higher-level services to collectives. These services usually involve those not available at the node level, such as CPU-intensive operations.
Super-collective	A collection of supernodes, acting in the same way as the collective to achieve resiliency among supernodes.
Zone	An area of the network or Internet under the authority of a super-collective. Facilitates management and administration, and achieves scalability.

3.3. Entities

We have designed a number of entities for the IDF, which are summarized in Table 1. These entities can be thought of as the basic building blocks of the IDF with simple functions. They will be used as the foundation for larger components with more advanced functions, which will be described later in the paper. An outline of these entities is shown in Fig. 1.

3.3.1. Nodes and supernodes. We would first like to introduce the concept of a node in the context of the IDF. The node is the most primitive entity in the IDF architecture. Its main function is to exchange messages securely with other nodes, as part of the solution to the information sharing requirement. Apart from the main function, the node also performs other tasks which work on the shared information. These other tasks will be discussed in detail in Section 4.

A node consists of two parts: a host and a software agent. A host is a computing device, whether it is a workstation, mobile device, laptop computer, and so on. A software agent is a continually running program on the host that performs the tasks required for the IDF.

There are two types of nodes: a normal node (as described in the previous paragraph) and a supernode. A supernode is a special node that performs higher-level functions compared to a normal node. The main purpose of a supernode is to provide a location for data analysis.

In terms of the two-level hierarchical model proposed in Section 3.2, nodes reside at the lower level while supernodes reside at the higher level in the hierarchy. In Fig. 1, the nodes are represented by the white boxes while the supernodes are represented by black boxes. Without going into the details, the basic idea is that the node will register itself with the supernode to obtain services. For example, a node may collect data on its host and forward the data to the supernode for analysis.

3.3.2. Collectives and super-collectives. In order for the IDF to be survivable, nodes and supernodes by themselves are not enough. If a node or a supernode fails, the data and analysis results on it may be lost forever. To circumvent this, we introduce two new entities: the collective and super-collective. A collective is a collection of nodes using the topology of a ring (see Fig. 1). Likewise, a super-collective is a collection of supernodes (a super-collective is also known as a supernode collective).

The reason it is in the form of a ring is to achieve data availability and survivability through data replication. Similar to the way peer-to-peer file sharing systems work, the nodes in a collective can exchange data to introduce redundancy. This minimizes the loss of data in the event of a crash or denial-of-service attack.

3.3.3. Zones. A zone is a logical area of the IDF that is placed under the authority of a super-collective. Zones simplify administration and management of the super-collective and their associated collectives. The concept of a zone is similar to that of a subnet in a LAN; just like how a LAN can be segmented into smaller subnets for easier administration, a zone can be used to divide a large organization down into smaller more-manageable areas. Having the concept of zones also allow scalability and compartmentalization of information.

The use of a zone can be explained better if an example is used. Suppose the IDF is used at a university. If it is a

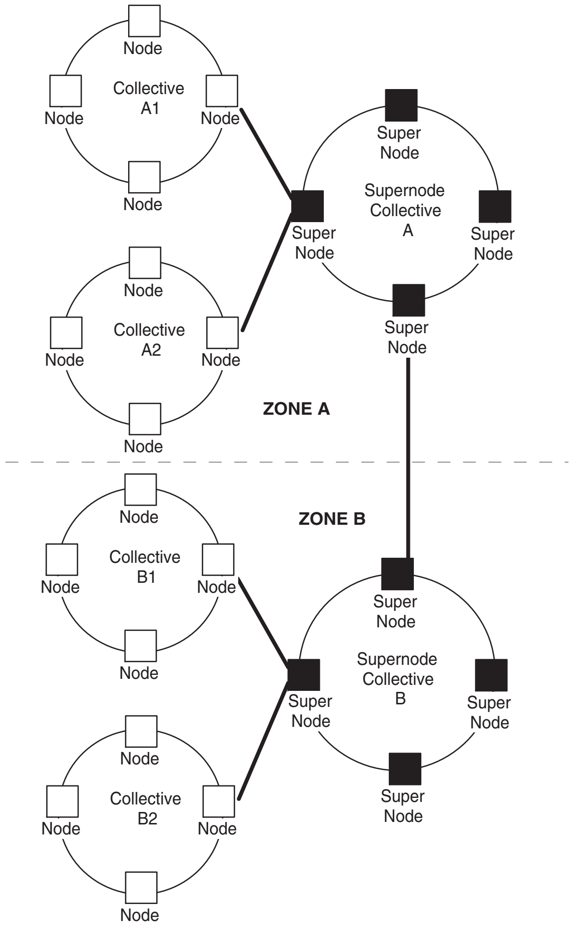

Figure 1. The entities in the IDF architecture

large university, a zone can be set up for each department. If needed, a large department can be segmented into even smaller zones, say, by research units. Using zones, the administrator can scale the IDF according to the size of the organization. Likewise, a company can have separate zones for its branches and departments.

3.3.4. Summary of entities. Since it is very important to understand the concept of the IDF entities before continuing with the rest of the paper, we provide a summary here. The basic entity of the IDF is a node. Nodes form collectives. A special node called the supernode forms super-collectives. Collectives register themselves with super-collectives. The area under the authority of a super-collective is called a zone. Nodes enable information sharing. Supernodes provide a location for analysis. Collectives allow redundancy to support security and survivability. Zones facilitate management and administration, as well as scalability.

4. Major components

In order for the entities (nodes, supernodes, etc.) described in the previous section to fully support the requirements outlined in Section 2, the entities have to be used to carry out more functions. For example, the nodes and supernodes have to collect data, perform analysis, respond to events, among other functions. To implement these higher-level functions, more specific IDF components that carry them out have to be built. We can think of the nodes and supernodes as basic building blocks, and the components described in this section are built on top of them.

If we compare this to the Internet, nodes and collectives would be similar to hosts and networks. The components here would be similar to the applications of the Internet when the hosts and networks are chained together. For example, applications like the World Wide Web and email services require the hosts and networks to be in place for them to work.

We propose eight such components to perform various high-level IDF functions. The first two components are required for all other components to work. They are the IDF Adaptation Layer and the communication and recovery subsystem. The next three components (data collection sensor, analysis engine, and response engine) provide the core functions of the IDF for intrusion detection. The last three components provide other services for the IDF.

This section describes each component of the IDF. For each component, we explain its purpose, why it is needed, and how it fulfills the requirements presented earlier in the paper. We also explain the roles of both the node and supernode in carrying out the function of the component. Fig. 2 is a diagram showing how the components and nodes are related.

As a side note, it should be mentioned that not all components are automated. Some functions require some human intervention for them to work. It should also be noted that these components are very large and they carry out many functions. It is not possible to explain all the concepts and functions of each component in this paper alone. Therefore, we will just discuss the basic high-level concepts of each component, and how they interact with the nodes and supernodes. The specific functions of the components themselves will be the subject of future papers.

4.1. IDF Adaptation Layer (IDFAL)

One of the most important components of the IDF is the Intrusion Detection Force Adaptation Layer (IDFAL) (Fig. 3). The IDFAL is an integration component that interfaces with the host's existing security software (other IDSs, firewalls, anti-virus software), applications, and the operating system. It provides a layer of abstraction between

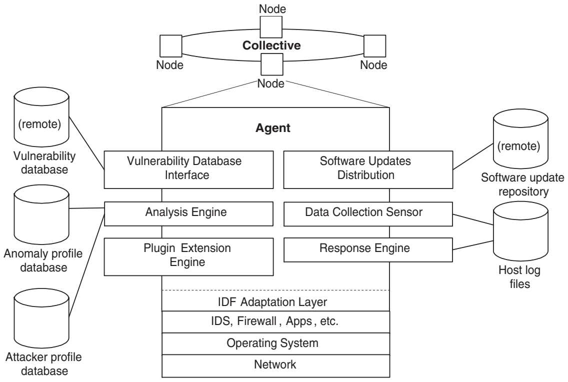

Figure 2. How the components relate to the nodes. Each node performs a function of every component. The actual function performed is different depending on whether the node is a regular node or supernode.

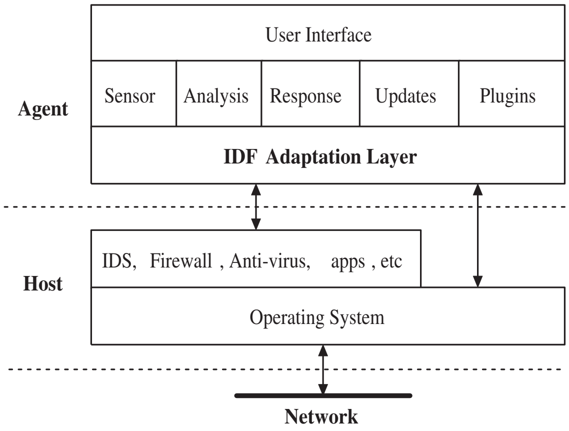

Figure 3. The IDF Adaptation Layer

platform-specific parts and the IDF components.

While the IDFAL has a few similarities with virtual machines, such as Java's, the main difference is that the IDFAL is not a separate entity that interprets and executes code given to it. We would liken it more to the system call API of an operating system that provides platform-specific low-level functions. This means that the IDFAL is precompiled into the node itself (or more specifically, into the IDF software agent running on the host). An advantage of this approach is that it achieves higher speed and better efficiency compared to the separately-compiled virtual machine-style approach.

The IDFAL provides a generic platform- and network-independent interface to the higher-level IDF components. Note that while we mention network independence, we believe it would be more efficient to engineer the IDFAL to be partially specific to TCP/IP, since it is the predominant protocol suite being used on the Internet today and in the foreseeable future. Welz and Hutchison [26] have done some work on interfacing trusted applications with IDSs, but their work focuses on tight coupling of IDSs. The IDFAL is designed to be modular and aims to provide a more loosely-coupled separate layer of interaction in order to achieve platform independence as well as support tasks not related to intrusion detection, such as updates and plugins.

To gain a better idea of how the IDFAL works, consider the case when an IDF node is deployed into a heterogeneous environment with different IDSs. The task of collecting data from the IDSs individually in such an environment would be difficult. With the IDFAL in place, the specific IDSs can be queried in a uniform manner by the higher-level IDF components. It is clear that the task of developing the IDFAL is not trivial. In order for the IDFAL to access operating system services, networks, and security products in an independent manner, and yet still take advantage of their different features, the IDFAL API has to be designed very well.

4.2. Communication and Recovery Subsystem

The Communication and Recovery Subsystem is used to handle communication and recovery operations. The reason that the functions of communications and recovery are combined together into one subsystem is because the two operations have to be tightly coupled for the purposes of the IDF. Communications should be handled concurrently with recovery to achieve the requirements of security and survivability. Likewise, recovery operations are highly dependent on the communication protocols.

The communications part of this subsystem has a few responsibilities at the node level. Firstly, it facilitates the exchange of messages between nodes. This includes node-to-node, node-to-supernode, supernode-to-node, and supernode-to-supernode communications. For collective-to-collective communication, the communication subsystem automatically elects a node to be in forwarding mode to represent the collective. If the node fails, a new forwarding node is automatically re-elected (this is actually part of the recovery process). Secondly, it is responsible for all aspects of node addressing and routing. Node addresses are assigned and identified using this subsystem. If a node wishes to communicate with another node, the subsystem on the source node queries neighboring nodes to work out the best route to the destination node. This can be complicated, as the task of developing an efficient routing protocol on a new type of network is not trivial. Thirdly, the communication subsystem addresses congestion issues. This responsibility is especially important when the underlying physical network is unreliable, such as in wireless networks. Lastly, it handles registration of nodes with supernodes.

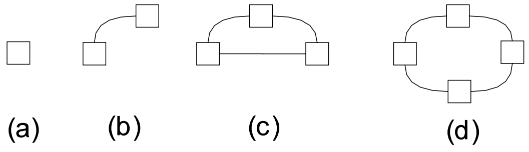

Figure 4. Formation of a collective

At the collective level, the communication part of the subsystem initiates the formation of a new collective and keeps track of changes in the collective. For example, it is responsible for node additions and removals. It also needs to maintain the state of nodes in the collective (whether they are active, idle, unreachable, etc.), so that the nodes can carry out their message exchanges and make routing decisions accurately. While this is the desire of the design, this is not an easy problem to solve. In order for every node to know the states of other nodes, the communication subsystem has to either broadcast messages out to every node in the collective, or specify a policy that a node will only know the states of its neighboring nodes. Both approaches have their advantage and disadvantages. The first approach ensures that each node has an accurate picture of the states of the nodes, but it has the danger of flooding the network. The second approach imposes less burden on the network, but may not give timely accurate state information, thus affecting routing decisions. Finding the right balance between the two approaches will definitely be a challenge.

As an example of how the communication subsystem works, Fig. 4 shows a step-by-step process of a formation of a collective. The figure shows one node initiating a collective, and three subsequent nodes joining the collective one at a time. Communication messages will have to be exchanged among the nodes in order to form this four-node collective.

The recovery part of the subsystem handles operations such as data replication and assisting a collective to recover from node failures. Data replication is used to introduce redundancy to minimize the impact of data loss in the event of a node failure or denial of service attack. If a node does fail (whether due to an attack or hardware failure), the communication and recovery subsystem initiates reconstruction of a collective. One possible approach for this recovery operation can be seen in Fig. 5. In the diagram, a four-node collective (Fig. 5a) has a member that suddenly fails (Fig. 5b). The neighboring nodes of the failed node detect this failure and change their states, thereby forming the temporary collective in Fig. 5c. Suppose now that the failed node recovers. It will then communicate with its former neighbors and reinstate itself (Fig. 5d), which results in the reconstructed collective in Fig. 5e. This is just one approach that may be used; more sophisticated recovery schemes would be needed for more complex recovery scenarios.

4.3. Data Collection Sensor

The data collection sensor performs the task of gathering events from the host and network. Collected data are logged on the host's log files. These data are then sent to the analysis engine so that they can be used to identify intrusions as well as provide intelligence for future investigations on suspicious behavior. The ability of this component is specific to the IDS that we are wrapping around. Again, the reason why it wraps around other IDSs is because it is unlikely that users will replace their IDS systems that they have already deployed.

The functions provided at the node level are to log host-specific events and to forward collected data to supernodes at periodic intervals (so as to not overwhelm the supernodes). At the supernode level, the component collects data from nodes. This data are then analyzed at the supernode.

4.4. Analysis Engine

Implementing data analysis on the IDF produces a significant challenge, since there may be many hurdles to overcome when dealing with data processing over such a large scale. We believe that if the IDF were to achieve its full potential, it has to perform three types of tasks: signature-based intrusion detection, anomaly detection, and attacker profiling. The large scale of an IDF deployment would result in the generation of a large amount of data. As such, processing this data with both signature-based pattern analysis and anomaly detection techniques should yield interesting results. We also believe attacker profiling is important. Attacker profiling refers to the method of gathering unique behavioral signatures of an attacker, which will be useful for law enforcement at a later stage. Examples of a "unique behavioral signature" include the type of words an attacker uses to deface a website, and the set of frequent typos made by the attacker at the UNIX shell.

The analysis engine is deployed on two levels: at the node level, as well as at the supernode level. The node-level analysis is similar to what a normal IDS would do. The node-level analysis engine would perform audit reduction and lightweight analysis. It then forwards relevant analysis results to the supernode-level analysis engine. The supernode-level analysis engine performs higher-level processing which includes identifying trends and suspicious behavior found in the zone or other zones. It also performs audit reduction and shares its analysis results with neighboring super-collectives.

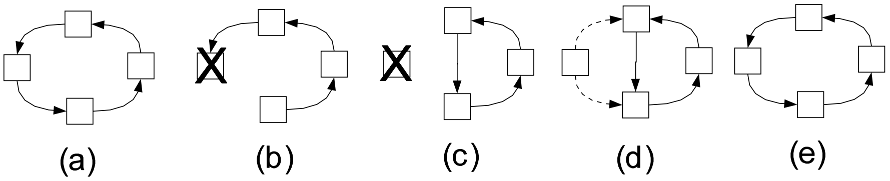

Figure 5. Reconstruction and recovery in a collective

Another difficult challenge that will be encountered when implementing the Analysis Engine is the need to synchronize the analysis results. How do we ensure that data and results in all nodes are always consistent? This difficulty warrants deeper investigation needed to develop a solution. Another problem, which is also related to the data collection sensor, is the issue of storage space. Being large scale and distributed, the IDF nodes will definitely produce a large amount of data to be processed. There will be issues in finding storage space for the data to be stored. One approach to address this would be to set up a quota for audit results. Since this may lead to lost results when the quota is filled, the results should be sent to another node with higher storage quota for safekeeping and redundancy purposes.

4.5. Response Engine

If intrusions are found, the IDF uses the Response Engine to send the results to the systems administrator. It provides a generic interface to the actual response mechanism, which may be to a log file, a window alert, pager, email, short message service (SMS), and so on. As described earlier, the response engine can work in both active and passive mode depending on the policy. The response engine gets its data from nodes and supernodes in various zones.

The response engine may also take proactive attack countermeasures to prevent further damage to the host, such as requesting new software updates for the host's platform to prevent future attacks using the same vulnerability. Another preventative measure is to dynamically change firewall rules on the host to ensure that future attacks of the same nature do not happen. Like the other components, the issues involved to enable these functions in the response engine are complex. The IDF has to allow these types of responses but at the same time adhere to the security policy.

4.6. Vulnerability Database Interface

The Vulnerability Database Interface (VDI) provides an interface to the IDF's own Vulnerability Database as well as publicly-known vulnerability databases. Examples of public vulnerability databases include the SecurityFocus vulnerability database [19] and CVE [24] (while defined as a "dictionary" instead of a database, the IDF should still be able to use CVE for this purpose). The idea of having the VDI is to provide up-to-date vulnerability information to the other components such as the analysis engine and software updates distribution engine. In the case of the CVE, it should be noted that since the CVE is primarily meant to be used by humans, there may be issues in automating this with the VDI.

4.7. Software Updates Distribution Engine

The Software Updates Distribution Engine is designed to enable the ability to preemptively react to an attack before it happens. The software updates distribution engine is responsible for propagating software updates to IDF nodes in a fast and timely manner. This engine also makes sure that the software updates are delivered in a secure manner. Software updates are stored and mirrored in many repositories to ensure that they are always available even when one repository fails.

For the software updates distribution engine to work, the node will first register software applications and operating systems and versions with its supernode. The supernode then pushes newly released and audited software updates to the registered nodes.

Software updates also have to be authenticated before they can be applied to the hosts. We have successfully prototyped the software updates distribution engine with favorable results [23].

4.8. Plugin Extension Engine

The Plugin Extension Engine is designed to fulfill our requirement of extensibility. The Plugin Extension Engine allows third-party developers to develop plugins for the IDF agent. This allows the IDF agent to perform tasks that it was not originally designed for, thereby supporting our design goal of extensibility. We believe that the scale of the IDF and its distributed architecture has the potential to allow innovative applications to be built on top of it. However, this ability may also be abused by malicious attackers who may develop malicious plugins for the IDF. As such, third-party IDF plugins may need to be authenticated and certified before they can be used.

4.9. Applications of the IDF components

When these IDF components work together, they can be used in many applications that help to enhance the security of the IDF community and the Internet. The following is a short list of such applications that can be achieved:

1. **Internet-scale intrusion detection.** The components can be used to achieve large-scale distributed intrusion detection throughout the Internet. Through distributed data collection via the sensors and data processing using the analysis engine in different zones, we can use the IDF to detect highly distributed and stealthy attacks.

2. **Proactive intrusion prevention.** Through proactive response and countermeasures based on attack information gathered from other zones and frequent software updates, the IDF can preemptively set up defenses to prevent intrusions and attacks before they occur.

3. **Policy enforcement.** The IDF can be set up to distribute nodes into different zones in a typical organization. This makes it convenient for enforcing policy. For example, when a new policy is introduced, an administrator can automatically ensure the policy is enforced on all the nodes in the zone. This is especially useful if every system in the organization is an IDF node. This capability also has the potential of reducing the workload and burden on the administrator.

4. **Trust management.** There has to be a form of trust management among the nodes, since inter-organizational information sharing is used in the IDF. IDF nodes also have the potential to enable trust management on their host systems too.

5. **Incident handling.** Incident handling is made more efficient through tools in the Analysis Engine like attacker profiling. This also assists law enforcement and computer forensic teams.

Apart from the short list of applications mentioned here, more interesting uses of the IDF will be realized as we examine and revise the IDF architecture in detail. Other future applications and uses that have not been presently realized can be introduced at a later stage through the Plugin Extension Engine.

5. Current implementation

Due to the size of the proposed IDF infrastructure, it is not possible to develop a full-scale implementation in the short term. Therefore, we have built a small research prototype of the architecture at present. Our initial goal was to use the prototype to test the feasibility of the IDF architecture in terms of the following test requirements:

1. **Interoperability.** How well does the prototype work with different platforms and architectures?

2. **Basic node-supernode communication.** How feasible is the node-supernode model? Is it sufficient for the actual IDF at a later stage?

3. **Response time.** How well does the model perform?

In order to gauge the effectiveness of the architecture using this prototype, we made the decision to build the prototype as a small-scale implementation of one of the major IDF components. The specific IDF component to use was another decision to consider. The component chosen must be quick and easy to prototype within the constraints of our time and resources. It must have simple but well-defined roles for the nodes and supernodes. It must also allow us to test platform independence issues as part of the interoperability test requirement. Lastly, it must also allow us to evaluate performance via the response time requirement.

After careful consideration, we selected the Software Updates Distribution Engine as the component for prototyping. The updates engine is small enough to prototype, and is not so complex that it will be a hurdle for us to produce short-term results. It is easy to define roles for the supernodes and nodes – the nodes will register themselves with the supernodes, and when software updates are available, the supernodes will deliver them to the nodes. We can measure the delivery time and software update application time to evaluate performance. As a bonus, it also lets us test interoperability issues since it is easy to prototype and therefore developing it in a platform-independent manner is not difficult. In contrast, compare this to the tasks required if we decided to implement the data collection sensor instead. For our tests to be worthwhile, we would also have to implement the analysis engine and response engine, resulting in a significant increase in complexity.

The software updates distribution engine prototype was developed using ANSI C++ (for portability and speed) on the Linux and OpenBSD platforms. It was tested on a small Ethernet network with five hosts as shown in Fig. 6. Two Linux distributions and OpenBSD were used at the nodes to fulfill the interoperability requirement (Table 2). Yet another Linux distribution was used at the supernode. XML was used to represent the hosts' system configurations, and

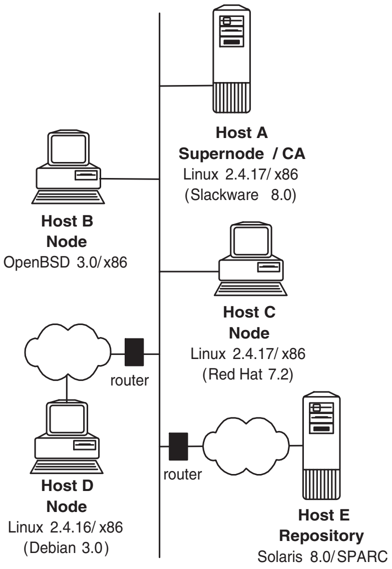

Figure 6. The test network used to test our first experimental prototype

Table 2. Hosts in the test network

Host	Role	OS	Architecture
A	Supernode/CA	Linux 2.4.17 Slackware 8.0	x86
B	Node (Webserver)	OpenBSD 3.0	x86
C	Node (Desktop)	Linux 2.4.17 Red Hat 7.2	x86
D	Node (Laptop)	Linux 2.4.16 Debian 3.0	x86
E	Repository	Solaris 8.0	SPARC

SSL was used for authentication and confidentiality of communication among the nodes.

Host A was set up to simulate a supernode collective and also to double as a Certificate Authority (CA) for signing and issuing certificates for SSL. Host E was set up as a software updates repository, which was configured as a web server to deliver updates using the HTTP protocol. Hosts B, C, and D acted as the nodes and were issued signed certificates by the CA as part of the initial preparation.

The prototype was tested using three UNIX programs – sudo (for OpenBSD), pine (for Red Hat), and wu-ftpd (for Debian). The supernode (Host A) was configured with a database that maps the software update for the UNIX program with its associated operating system. For example, Red Hat Linux was mapped to pine. This means that the pine version on the Red Hat machine is vulnerable, and should be replaced. The sudo and pine packages we used are official packages from Red Hat and Debian respectively, while the sudo update was custom-made for OpenBSD.

In our tests, the nodes gathered information about their system configuration (operating system name, version, etc.) and registered them with the supernode over an SSL connection. We then simulated the scenario when a new update is available. The supernode checks its updates database with those of the registered nodes configuration, and if there are matches (for example, Red Hat and pine, as they were mapped in the database earlier), the appropriate update was delivered to the node and applied after an integrity check. Table 3 shows the download speed ($S1$) and installation speed $S2$ (including integrity check time) for the updates, the hosts' CPU speed, update method (M), and update size. The update installation speed $S2$ should be read relative to the CPU speed and update size. The pine update application process took 19.76s because of its large size (2.63MB) and it was run on a slow CPU (166MHz). The software updates distribution engine, the prototype, and these results are explained in greater detail in our earlier work [23].

Table 3. Updates applied

Update	M	CPU	Size	$S1$	$S2$
sudo 1.6.5	custom	400MHz (Host B)	71.2KB	0.06s	0.06s
pine 4.44	rpm	166MHz (Host C)	2632KB	1.55s	19.76s
wu-ftpd 2.6.1	dpkg	700MHz (Host D)	250KB	0.24s	4.1s

5.1. Ongoing and future implementation

We are now in the process of rebuilding the test network for future IDF development. To test interoperability, we are upgrading the test systems and adding new machine architectures, operating systems, firewalls, and intrusion detection systems into the network. To test scalability issues such as the feasibility of scaling from a small number to a large number of nodes (as mentioned in Section 2.2), we will simulate collectives and super-collectives using multiple virtual machines running on VMware [2]. We are also extending the test network into the wireless arena, with particular interest in Bluetooth [1] and IEEE 802.11b.

We have already started the design of the other components. As the next stage of our research, we intend to continue developing the communication and recovery subsystem, as well as the Intrusion Detection Force Adaptation Layer (IDFAL). Once these two foundational components are ready, we will then develop the core components consisting of the data collection sensor, analysis engine, and response engine. Next, we will revisit the software updates distribution engine and start development of the vulnerability database interface and plugin extension engine.

As we proceed with the design and implementation of the IDF components, one of the most difficult problems we face is the identification of the information that should be exchanged among the IDF nodes. As stated earlier in Section 2, we have to consider requirements such as the organizational need for data privacy but at the same time, allow the information to be useful. It would be pointless to exchange information that is unnecessary, as this would result in useless alerts. This has been a significant challenge since the beginning of the IDF project, and we are working on methods to address it.

Another interesting challenge has to do with the fact that the IDF is intended to be survivable. The IDF will undoubtedly face attacks against itself, and in fact, this is expected. A successful attack against the IDF from the Internet may affect IDF communications and responses. With this in mind, it would be worth considering out-of-band communication methods which are non-Internet-based, thus providing alternative communication avenues and making the IDF less susceptible to such attacks.

6. Related work

The architecture and design of the IDF were inspired by many technologies, ranging from distributed intrusion detection systems to peer-to-peer systems. The distributed intrusion detection systems include DIDS [20], which uses the paradigm of distributed data collection and centralized data processing. CSM [27] uses a true peer-based and totally decentralized approach. Another influential DIDS was AAFID [4], which is an agent-based DIDS which uses a three-level hierarchy. Low-level agents perform data collection and low-level analysis. The results are sent to transceivers, which in turn analyze and pass their results to monitors, the highest-level entity in the hierarchy. AAFID achieves scalability by allowing monitors to be "hooked" to one another, thus creating a DIDS. AAFID is no longer supported by its authors. Other influential IDSs include EMERALD [18] and GrIDS [21], and intrusion detection research performed by Kruegel and Toth [14], and Vigna, Kemmerer, and Blix [25].

Apart from IDSs, the IDF architecture also draws some ideas from Internet-Scale Operating Systems (ISOSs) such as Chord [22], OceanStore [15], and Tapestry [28]. Peer-to-peer technologies [17] like Gnutella, distributed.net, and Freenet [8] also influenced a few of our design decisions.

7. Conclusion

We described the architecture and design of an Internet-scale Intrusion Detection Force (IDF), which is a virtual infrastructure on top of the current Internet. Its main goal is to defend organizations and protect the Internet as a whole, through secure inter-organizational information sharing and intelligent distributed data analysis and response. We believe this infrastructure has many applications, the most significant of which is its ability to detect, prevent, and respond to highly distributed and stealthy attacks.

Acknowledgments

We would like to thank the anonymous reviewers of this paper for their helpful comments and suggestions.

References

[1] The Official Bluetooth Website. http://www.bluetooth.com/.

[2] VMware. http://www.vmware.com/.

[3] S. Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Dept. of Computer Engineering, Chalmers University of Technology, Sweden, March 2000.

[4] J. S. Balasubramaniyan, J. O. G. Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. Technical Report 98/05, COAST Laboratory, Purdue University, May 1998.

[5] A. Beguelin, E. Seligman, and P. Stephan. Application level fault tolerance in heterogeneous networks of workstations. *Journal of Parallel and Distributed Computing*, 43(2):147–155, 1997.

[6] S. M. Bellovin. Security weaknesses in the TCP/IP protocol suite. *Computer Communications Review*, 2(19):32–48, 1989.

[7] S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, J. Row, S. Staniford-Chen, R. Yip, and D. Zerkle. The design of GrIDS: A graph-based intrusion detection system. Technical report, Department of Computer Science, University of California at Davis, January 1999.

[8] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009*, 2001.

[9] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. A. Longstaff, and N. R. Mead. Survivability: Protecting your critical systems. In *Proceedings of the International Conference on Requirements Engineering*, Colorado Springs, April 1998.

[10] J. Haines, V. Lakamraju, I. Koren, and C. M. Krishna. Application-level fault tolerance as a complement to system-level fault tolerance. *The Journal of Supercomputing*, 16:53–68, 2000.

[11] J. W. Haines. Application-level fault tolerance. Master's thesis, University of Massachusetts Amherst, May 1999.

[12] J. Hochberg, K. Jackson, C. Stallings, J. F. McClary, D. DuBois, and J. Ford. NADIR: An automated system for detecting network intrusion and misuse. *Computers and Security*, 12(3):235–248, May 1993.

[13] Internet Engineering Task Force. Intrusion Detection Exchange Format (idwg) charter. http://www.ietf.org/html.charters/idwg-charter.html.

[14] C. Kruegel and T. Toth. Distributed pattern detection for intrusion detection. In *Proceedings of the Network and Distributed System Security*, San Diego, CA, February 2002.

[15] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, and H. Weatherspoon. OceanStore: An architecture for global-scale persistent storage. In *Proceedings of Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, November 2000.

[16] P. Mockapetris. Domain names: Concepts and facilities, RFC 1034, November 1987.

[17] A. Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[18] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the 20th National Information System Security Conference*. National Institute of Standards and Technology, October 1997.

[19] SecurityFocus. SecurityFocus Vulnerability Database. http://www.securityfocus.com/corporate/products/vdb.

[20] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C.-L. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur. DIDS (Distributed Intrusion Detection System) - motivation, architecture, and an early prototype. In *Proceedings of the 14th National Computer Security Conference*, pages 167–176, October 1991.

[21] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS a graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Security Conference*, Baltimore, MD, October 1996.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM'01*, San Diego, CA, 2001.

[23] L. Teo and Y. Zheng. Secure and automated software updates across organizational boundaries. In *Proceedings of the 2002 IEEE Workshop on Information Assurance*, pages 212–219, United States Military Academy, West Point, NY, June 2002.

[24] The MITRE Corporation. Common Vulnerabilities and Exposures (CVE). http://cve.mitre.org/.

[25] G. Vigna, R. A. Kemmerer, and P. Blix. Designing a web of highly-configurable intrusion detection sensors. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, pages 69–84, Davis, CA, October 2001.

[26] M. Welz and A. Hutchison. Interfacing trusted applications with intrusion detection systems. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, pages 37–52, Davis, CA, October 2001.

[27] G. B. White, E. A. Fisch, and U. W. Pooch. Cooperating security managers: A peer-based intrusion detection system. *IEEE Network*, 10(1):20–23, January/February 1996.

[28] B. Y. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UCB, 2001.